Vectorized Method for Solving the n-queens Problem using Bohrium

Larsen, Mads Ohm

Publication date:
2016

Document version
Publisher's PDF, also known as Version of record

UNIVERSITY OF COPENHAGEN
NIELS BOHR INSTITUTE

Vectorized Method for Solving the n-queens Problem using Bohrium

Mads Ohm Larsen <ohm@nbi.ku.dk>

Introduction

On a normal 8×8-chessboard we find that 8 queens can be positioned in

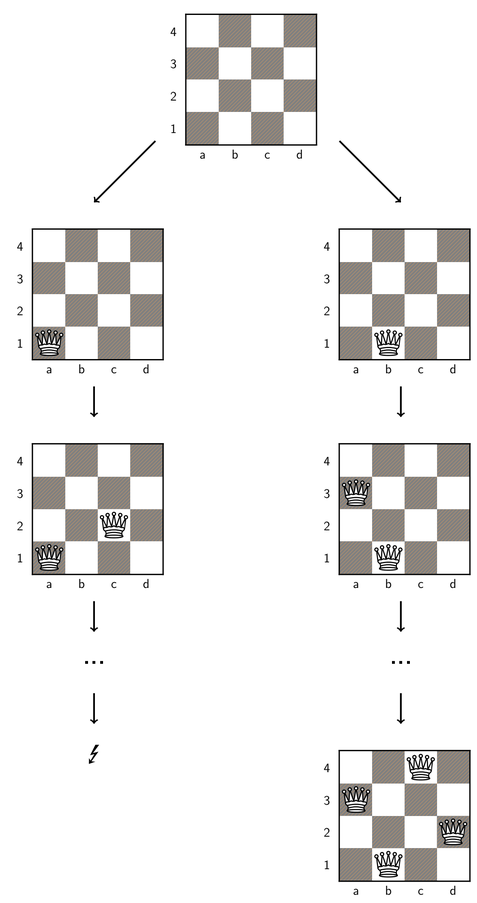
$$\binom{64}{8} = \frac{64!}{8! \cdot (64-8)!} = 4,426,165,368$$

different legal ways. This is too many to brute force, especially as we go for larger n.

For the 8×8-queens problem we only have 92 distinct solutions.

Backtracking Algorithm

Since only one queen are allowed in each row and column, we can skip placing another queen in the same column or row in the backtracking algorithm.



The problem is simple, but non-trivial to solve. Using the knowledge stated before, we *only* have to check

$$8^8 = 2^{24} = 16,777,216$$

placements, instead of the roughly four and a half billion legal placements.

Using Permutations

Instead of looking for the right combinations, we can instead compute permutations of rows, with a queen in each column. There are

$$8! = 40,320$$

permutations of a 8×8-chessboard's rows.

Because we construct the rows with a queen in each column, we now know we only have to check the diagonals, to see if the current permutation is a solution.

Figure 1 shows the "identity" chessboard, with a queen in each row and column.

It is of course not a solution, but permuting the rows will eventually grant the 92 solutions.

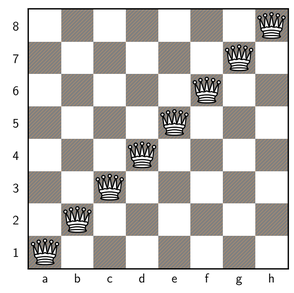


Figure 1: The 8×8 "identity" chessboard.

If we do the same calculations for $n = 4$ as we did for $n = 8$, we get that we have

$$\binom{16}{4} = 1,820$$

legal configurations of the board, which can be boiled down to

$$4^4 = 2^8 = 256$$

placements where we only check the diagonals, but only $4! = 24$ permutations of rows.

The 4×4-"identity" chessboard, which is shown in figure 2, can be built similar to figure 1.

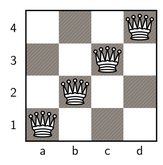


Figure 2: The 4×4 "identity" chessboard.

For the four-queens problem there are only two solutions. These are shown in figure 3.

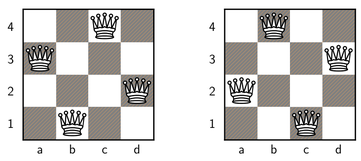


Figure 3: The 2 solutions for the four-queens problem

These two are reflections (Figure 4) or rotations (Figure 5) of each other.

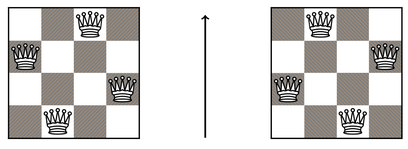


Figure 4: Reflection

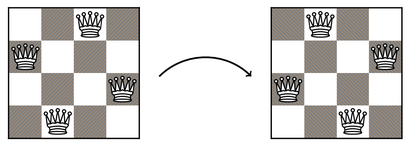


Figure 5: Rotation

Normally we count two different types of solutions, *distinct* and *fundamental*. The distinct solutions do not take reflection and rotation into consideration, but the fundamental solutions do. For 8×8 we have 92 distinct solutions, but only 12 fundamental.

Matrix Representation

One solution for the four-queens problem can be represented as the following matrix:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

To check if this is a solution, we can sum along each column, row, and diagonal, checking if the sum of any of these are greater than 1. If not, then we have a solution.

Gathering all the traces (the sums along the diagonals and offset-diagonals) for the 4×4 identity board we get

$$[0, 0, 0, 4, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1]$$

Here we see a 4, which means that the identity board isn't a solution to the four-queens problem. In fact, the maximum trace must be equal to 1 to be a solution.

Adding a Dimension

Now that we know how to solve one board, we can create all permutations of that board, to solve the entire n-queens problem for some n.

We can do this, by adding a dimension to our matrices above, creating a tensor of 4×4-chessboards.

Doing so yields

$$4 \cdot 4 \cdot 4! = 384$$

boards, which all needs to be checked. Generalizing for the n-queens problem, this would be

$$n^2 \cdot n!$$

For the four-queens problem, this yields $4! = 24$ traces of which we only need to count the number of 1s. As we have seen already, there is only two solutions, and we only have two max-traces which are 1.

Note that since there is always queens in the diagonals or offset-diagonals, we can never have a max-trace of zero value.

Using NumPy and Bohrium

With our knowledge, we can construct a simple Python program using NumPy or Bohrium to calculate how many distinct solutions are to the n-queens problem for a given n. In this program we simply list all permutations of the $n \times n$-chessboard, flip it, than trace all diagonals from $-n$ to n and max that matrix into a 1D vector. The result is then found by counting 1s in this vector.

Future Work

Unfortunately we do not currently gain any performance boost using Bohrium, however this is largely due to the creation of the permutation boards.

We are planning to implement a permutation generator into the Bohrium runtime as a streaming generator for the GPU, and will hopefully get a performance speed-up doing so.

```python
import numpy as np
# import bohrium as np
from itertools import permutations

def nqueens(n):
    # Generate all permutations of n*n identity boards
    boards = list(permutations(np.eye(n)))

    # Attach the flipped boards as another dimension
    rotation_boards = np.array([boards, np.fliplr(boards)])

    # Calculate all the traces, from -n to n for all the boards
    m = np.max([
        np.trace(
            rotation_boards,
            i,
            axis1=2,
            axis2=3
        ) for i in xrange(-n, n)
    ], axis=(0, 1))

    # Count the number of 1s in the maximum of the traces
    print np.sum(m[m == 1]), "solutions for", n, "by", n, "board."

nqueens(8) # => 92 solutions for 8 by 8 board.
```